

Nro. ord	Apellido y nombre	L.U.	Turno	#hojas
22	GIOKOR PALAZZINI TOMÁS AGUSTÍN	795/23	T	4

SISTEMAS DIGITALES - Segundo Recuperatorio
Segundo Cuatrimestre 2024

Ej. 1	Ej. 2	Ej. 3	Ej. 4	Nota
1	2	4	2	9

Correctorx:

Aclaraciones

- Anote apellido, nombre, LU y numere *todas* las hojas entregadas, entregando los distintos ejercicios en hojas separadas.
- El parcial **no es a libro abierto** pero pueden consultar la hoja de referencia provista por la cátedra.
- Justifique sus respuestas explicando lo que considere necesario en lenguaje natural.
- El parcial se aprueba con 6 y se deben tener ambos parciales aprobados para aprobar la materia (promoción directa).

Ejercicio 1 (2 pts.) Implemente las funciones definidas en el siguiente bloque de código C en lenguaje ensamblador de RISC-V.

```
int function cantidad_divisores(int k){
    if(k <= 1) return 1;
    return cantidad_divisores_rec(k, k-1);
}

int function cantidad_divisores_rec(int k, int n) {
    if (n == 1) return 1;
    int cantidad = cantidad_divisores_rec(k, n - 1);
    if(k % n == 0){
        cantidad = cantidad + 1;
    }
    return cantidad;
}

int es_primo(int k){
    return cantidad_divisores(k) == 1;
}
```

La función **es_primo** llama a **cantidad_divisores** para devolver 1 si la cantidad de divisores es igual a 1 y 0 en caso contrario. Un número es primo si es divisible (la división entera no produce resto) solamente por el mismo número y por uno. Debe respetar la estructura de llamadas entre funciones incluyendo la implementación recursiva. Recomendamos concentrarse en la traducción a código RISC V que respete la convención de llamada. Deben explicar en qué registros se almacenan los valores en cada paso y cómo se aseguran que se respeta la convención.

Ejercicio 2 (2 pts.) Implemente las siguientes funciones en lenguaje ensamblador de RISC V respetando la convención de llamada presentada en la materia. Describir el comportamiento y cómo se aseguran que se respete la convención.

- **int es_par(int x) = $x \% 2 == 0$** (chequeo de paridad)
- **void arreglo_par(int arr[], int largo):** Dado un puntero a un arreglo de enteros de 32 bits y la cantidad de elementos, cambia cada valor del arreglo por un 1 si el elemento era par y por un 0 en caso contrario, deben hacer uso de la función **es_par**.

Ejercicio 3 (4 pts.) Se tiene una estructura **BalanceDeudor** que contiene el ID del cliente como un entero con signo de 8 bits, la suma de sus consumos como un entero sin signo de 32 bits, la cantidad de pagos realizados como un entero sin signo de 16 bits, la suma de sus pagos realizados como un entero en complemento a dos de 16 bits. Ubicación de los datos de una estructura **BalanceDeudor**:

Byte	0x0000	0x0001	0x0005	0x0007
Nombre	ID	Consumos	Cant. pagos	Pagos

En memoria se encuentra un arreglo **balanceDeudores** del tipo **BalanceDeudor** con la forma:

Direccion	0x000	0x001	0x005	0x007	0x009	...	0x030	0x031	0x035	0x037	0x039
Valor	17	30020	2	-1232	6	...	9	5878	10	300	0

Donde el final del arreglo es demarcado por un ID nulo. Se pide

- Calcular cuántos bytes ocupa en memoria la estructura BalanceDeudor y cuántos bytes un arreglo de tipo BalanceDeudor de 32 deudores.
- Escribir una función contarDeudores(balanceDeudores) que dada una posición de memoria que indica el comienzo de un arreglo balanceDeudores de BalanceDeudor, devuelva un entero que indique para cuántas estructuras del arreglo vale que la suma de los consumos (segundo elemento de la estructura) es mayor que la suma de los pagos realizados (cuarto elemento de la estructura). Ejemplo:

```
.data
balanceDeudores: .byte 17
                  .word 30020
                  .half 10
                  .half 1232
                  .byte 6
                  .word 200
                  .half 200
                  .half 200
                  .byte 9
                  .word 5878
                  .half 58
                  .half 300
                  .byte 0      #Declaramos el final del arreglo

.text
contarDeudores:
    ...
```

Para este caso balanceDeudores debe devolver 2 ya que el usuario con id 17 y el usuario con id 9 tienen consumos mayores a sus pagos. Recuerden que el arreglo es pasado como dirección de memoria de su primer elemento a través del primer parámetro de la función.

Ejercicio 4 (2 pts.) Para una microarquitectura de ciclo simple para un procesador de RISC V, como la que vimos en clase, que debe ejecutar la instrucción `or x4, x5, x6`, ¿qué sucede si `ResultSrc` se encuentra siempre en 1? ¿Y si los bits 24 a 20 de la instrucción están en cero?

ORDEN 22

GIORGI PALAZZINI

LU 495/23

HOJA 2/4

TOMÁS AGUSTÍN

(2)

ES_PAR: } TOMA EL VALOR DE AO
 ANDI TO AO 1 } Y DEVUELVE 1 SI ES PAR
 XORI AO TO 1 } O CERO SI ES IMPAR
 JR RA

ARREGLO_PAR → HABIENDO CARGADO EL ARREGLO EN AO Y LA CANTIDAD EN A1 ✓

ADDI SP SP-8 ✓

SW RA 0(SP) ✓

LW TO 0(AO) ✓

SW TO 4(SP) ✓

ADDI A1 A1-1 ✓

BEQ ZERO A1 FINCICLO

ADDI AO AO 4

JAL ARREGLO_PAR

MV SO AO

LW AO 4(SP)

JAL ES_PAR

SW AO 0(SO)

ADDI SO SO-4

MV AO SO

LW RA 0(SP)

ADDI SP SP 8 ✓

JR RA

RESERVO ESPACIO EN EL STACK POINTER, LE CARGO EL RA Y EL ELEMENTO DEL ARREGLO, ME FIZO SI HAY ELEMENTOS TODAVÍA, ACTUALIZO LA DIRECCIÓN AL PRÓXIMO ELEMENTO Y REPITO HASTA QUE NO HAYA MÁS.

GUARDO LA DIRECCIÓN EN SO, CARGO EL ELEMENTO GUARDADO EN EL SP A AO, VEO SI ES PAR Y AÑADO EL RESULTADO A LA DIRECCIÓN EN SO, VOY AL ELEMENTO ANTERIOR, PASO LA DIRECCIÓN A AO, CARGO EL RA DEL SP, LO ACTUALIZO Y VOY AL RA.

CUANDO TERMINA DE EJECUTARSE QUEDA EN AO LA DIRECCIÓN ~~INICIAL~~ AL 1ER ELTO. D
 ARREGLO CON LOS ELEMENTOS ACTUALIZADOS

FINCICLO:

JR RA

ORDEN 22

GIORGI PALAZZINI

LU 995/23

HOJA 3/4

TOMÁS AGUSTÍN

③

LA ESTRUCTURA BALANCE DEUDOR OCUPA 9 BYTES, UN ARREGLO DE TIPO BALANCE DEUDOR CON 32 DEUDORES OCUPA 289 BYTES ($288 = 9 \cdot 32 + 1 \text{ BYTE NULO}$)

CONTAR DEUDORES

LB TO 0 (AO)

BEQ TO ZERO FINCICLO

ADDI SP SP -12

SW RA 0(SP)

LW TO 1(AO)

SW TO 4(SP)

LW TO 7(AO)

SH TO 8(SP)

ADDI AO AO 9

JAL CONTAR DEUDORES

LW TO 4(SP)

LH T1 8(SP)

BLT T1 TO SUMADOR

LW RA 0(SP)

ADDI SP SP 12

JR RA

SUMADOR:

ADDI AO AO 1

LW RA 0(SP)

ADDI SP SP 12

JR RA

FINCICLO:

ADDI AO ZERO 0

JR RA

CARGO EN AO EL ARREGLO

RESERVA

VEO SI EL BIT ES NULO, SI NO LO ES ~~REPORTA~~ MEMORIA EN EL SP
CARGO EL RA, LOS CONSUMOS Y LOS PAGOS. REPITO HASTA LLEGAR
AL ID NULO Y AHÍ ~~ACTUALIZO~~ EN CARGO CERO EN AO PARA
INICIALIZAR EL CONTADOR

CARGO LOS CONSUMOS Y LOS PAGOS EN TO Y T1 RESPECTIVAMENTE, SI T1 ES MENOR A T2 ~~NO~~ SUMO AL CONTADOR, CARGO EN EL
RA EL RA GUARDADO EN EL SP, ~~Y~~ ACTUALIZO EL SP Y RETORNO
A LA POSICIÓN DEL ~~SP~~ RA.

ORDEN 22

GIORGI PALAZZINI

LU 7/5/23

HOJA 4/4

TOMÁS AGUSTÍN

④ SI RESULT SRC ESTÁ SIEMPRE EN 1 ENTONCES EL MULTIPLEXOR QUE DECIDE ENTRE TOMAR EL DATO DE DATA MEMORY O LA ALU QUE CALCULA ENTRE LOS DOS REGISTROS FUENTE O EL REGISTRO Y EL INMEDIATO, DECIDE SIEMPRE TOMAR DE LA ALU

SI LOS BITS ENTRE 24 Y 20 SON TODOS CEROS, SE GUARDA EN EL DESTINO EL VALOR QUE HAY ~~EN EL REGISTRO QUE INDICAN LOS BITS ENTRE 19 Y 15~~ EN EL REGISTRO QUE INDICAN LOS BITS ENTRE 19 Y 15, YA QUE ES UN OR ENTRE UN VALOR Y CERO

B